

# **IBM PROJECT 2022**

## **INVENTORY MANAGEMENT SYSTEM FOR RETAILERS**

### **PROJECT REPORT DOCUMENTATION**

#### **TEAM ID: PNT2022TMID35555**

## **1. INTRODUCTION**

### **1. PROJECT OVERVIEW**

An inventory management system (IMS) is a computerized program that helps businesses track and manage their inventories. An IMS enables businesses to reduce the costs associated with inventory, and improve the accuracy and timeliness of supply. Businesses use an IMS for different types of goods including raw materials, supplies, and finished products. Most IMS programs include features for managing purchase orders, sales orders, and shipping and receiving. Many systems also include reporting capabilities such as sales and operations planning reports (S&OP) and monthly business reviews (MBR). Some systems are also designed for specific industries, such as healthcare and manufacturing. Retail IMS systems are geared toward small businesses with limited capital and resources and can automate the ordering and tracking of goods. Manufacturers and large wholesalers often purchase more robust IMS systems that are designed specifically for their needs.

An inventory management system is responsible for ensuring that the right quantity of the right product is available to customers at the right time and at the right price. A well-designed IMS will help improve efficiency, reduce costs, and minimize the risk of obsolescence or stockouts. Implementing an IMS can be a challenge; many businesses struggle to achieve effective inventory management without a dedicated resource to manage it. However, a solid plan and sound implementation strategies can help to ensure a successful outcome. The first step in implementing a new inventory management system is to understand the benefits and risks of the program.

## **2. PURPOSE**

The main purpose of an inventory management system is to help companies track the quantity, location, and condition of all inventory. This information can then be used to make decisions about where to allocate resources and when to order new products. Inventory management systems can also help companies reduce the amount of inventory they have on hand, which can save money and increase profits.

As your business grows, its inventory requirements will grow as well. Your inventory will become more complex, with items coming in from multiple suppliers and multiple warehouses. Managing your inventory manually will be challenging and time-consuming, making it difficult for you to maintain adequate levels of inventory to meet customer demand and grow your business.

## 2. LITERATURE SURVEY

### 2.1. EXISTING PROBLEM

One common problem that existed in most of the systems is the inability to track the inventory in real time. This is because the systems were not integrated with the point-of-sale system. This meant that the inventory was not updated in real time. This resulted in the loss of sales and profits.

### 2.2. REFERENCES

- [Research paper on Inventory management system](#)
- [Inventory management efficiency analysis: A case study of an SME company](#)
- [A Study of Inventory Management System - Case Study](#)
- [Informative Review on Inventory Control System](#)
- [Improvement of Inventory Management System Processes by an Automated Warehouse Management](#)
- [Study of Smart Inventory Management System Based on the Internet of Things \(IOT\)](#)
- [Research and Design of the Intelligent Inventory Management System Based on RFID](#)

### 2.3. PROBLEM STATEMENT DEFINITION

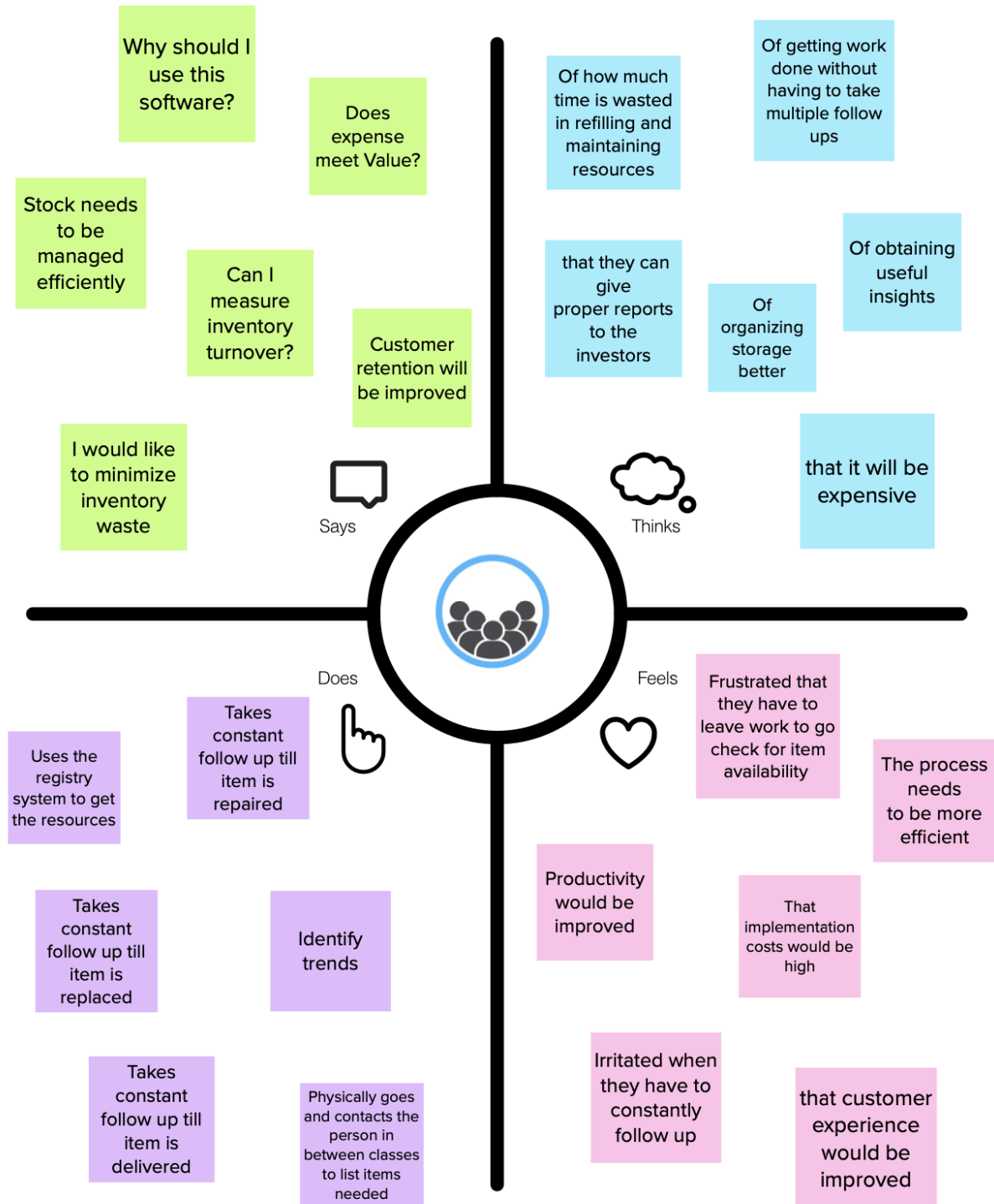
The main Objective of this Project is to provide a desktop based application that allows shops to monitor all IMS related information, including stock management, sales data and purchase information. The application enables retailers to manage their products flexibility and have complete insight into what is stored in their inventory, and request additional stock as and when needed.

<b>I am</b>	Retailers and Customers
<b>I'm trying to</b>	Have more insights on stocks and their availability to increase productivity
<b>But</b>	Manual management of the stocks are difficult and existing systems aren't much flexible
<b>Because</b>	Too much stock items cause bigger problems and current systems are obsolete

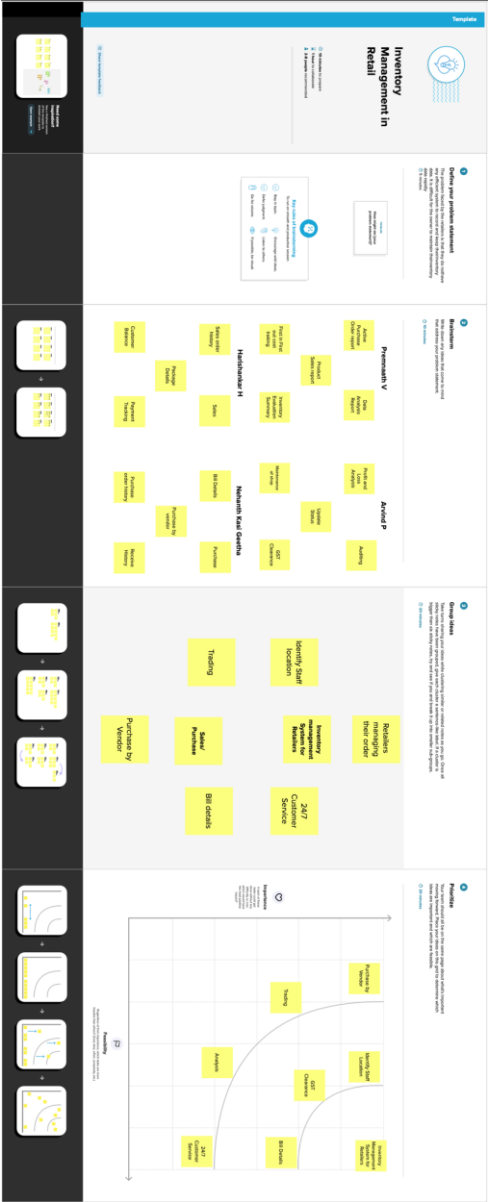
<b>Which makes me</b>	Want to create better inventory management system and increase the accuracy and flexibility of the vendors
-----------------------	--

### 3. IDEATION & PROPOSED SOLUTION

#### 3.1. EMPATHY MAP AND CANVAS



3.2. IDEATION & BRAINSTORMING



3.3.PROPOSED SOLUTION

1	Problem Statement (Problem to be solved)	The problem statement aims to make desktop application for retailers and to track all areas of Inventory Management System like purchase details, sales details, stock management and other policies.
---	---	---

2	Idea / Solution Description	<p>The application is developed to help retailers track and manage stocks related to their own products. The System will ask the retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application. Once retailers successfully log in to the application they can update their inventory details, also users will be able to add new stock by submitting essential details related to the stock. They can view details of the current inventory. The System will automatically send an email alert to the retailers if there is no stock found in their accounts. So that they can order new stock.</p>
3	Novelty / Uniqueness	<p>Apart from the standard features of the inventory management system like handling products, warehouses, locations we also plan to include the feature of sales prediction using</p>
		<p>regression and the previous sales data within our application.</p> <p>We also make the development and maintenance easier by containerizing the app using Docker</p>
4	Social Impact / Customer Satisfaction	<p>With this system we aim to make better use of the inventory available for the retailers. This improves the management and reduces excess inventory and thus reduces the wastage of products.</p> <p>It also improves the relationship with vendors and suppliers and can negotiate better deals with the suppliers by knowing the demand beforehand.</p>
5	Business Model (Revenue Satisfaction)	<p>Retailers can order the right amount and type of stock at the right time with the aid of an inventory management system. It eliminates the unnecessary expense for the retailers.</p>

6	Scalability of the Solution	A scalable cloud architecture is made possible through virtualization. Unlike physical machines whose resources and performance are relatively set, virtual machines (VMs) that we use in IBM cloud are highly flexible and can be easily scaled up or down. Kubernetes allows users to horizontally scale the total containers used based on the application requirements, which may change over time. It's easy to change the number via the command line
---	-----------------------------	---

3.4.

### 3.4. PROBLEM SOLUTION FIT

Define CS, fit into CC	<b>1. CUSTOMER SEGMENT(S)</b> <span>CS</span> Who is your customer? i.e. working parents of 0-5 y.o. kids  Retailers generally keep track of their merchandise from the time it is bought until it is sold.	<b>6. CUSTOMER CONSTRAINTS</b> <span>CC</span> What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices.  Openness to availability Network Restrictions Changing the cost of commodities Delays in delivery	<b>5. AVAILABLE SOLUTIONS</b> <span>AS</span> Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking.  usage of third-party inventory websites Management of log books in standard way Hiring employees and accountants to maintain stock	Explore AS, differentiate
	<b>2. JOBS-TO-BE-DONE / PROBLEMS</b> <span>J&amp;P</span> Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one, explore different sides.  Avoid overstocking Challenges in stock management Poor demand forecasting	<b>9. PROBLEM ROOT CAUSE</b> <span>RC</span> What is the real reason that this problem exists? What is the back story behind the need to do this job? i.e. customers have to do it because of the change in regulations.  Absence of real-time inventory control information	<b>7. BEHAVIOUR</b> <span>BE</span> What does your customer do to address the problem and get the job done? i.e. directly related, find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)  Information is essential for the creation and improvement of the application.	
Identify strong TR & EM	<b>3. TRIGGERS</b> <span>TR</span> What triggers customers to act? i.e. seeing their neighbour installing solar panels, reading about a more efficient solution in the news. Need separate knowledge for maintenance Maintaining large number of records by a single individual  <b>4. EMOTIONS: BEFORE / AFTER</b> <span>EM</span> How do customers feel when they face a problem or a job and afterwards? i.e. lost, insecure > confident, in control - use it in your communication strategy & design. Before - Worried, Frustrated, Lack of knowledge about stocks After - Happy, profitable, Flexible working	<b>10. YOUR SOLUTION</b> <span>SL</span> If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality. If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour. Development of an cloud application that "Tracks real-time inventory such as purchase details, sales information, and stock management" and "alters the user on less availability of Stock"	<b>8. CHANNELS OF BEHAVIOUR</b> <span>CH</span> <b>8.1 ONLINE</b> What kind of actions do customers take online? Extract online channels from #7 All inventory details available  <b>8.2 OFFLINE</b> What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development. SMS notifications for inventory	Extract online & offline CH of BE



Problem Solution fit canvas is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 license  
Created by Daria Nepriakhina / Amaltama.com





## 4. REQUIREMENT ANALYSIS

### 4.1 FUNCTIONAL REQUIREMENTS:

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registering through a form Registering through mail
FR-2	User Confirmation	Email confirmation OTP confirmation
FR-3	Login	Log in to the application by entering required credentials (email ID and password)
FR-4	Dashboard	View the products details (Name, quantity)
FR-5	Add items to the Inventory list	Users can add items that they wish to buy to the inventory
FR-6	Stock Updation	Increasing the availability of a particular product

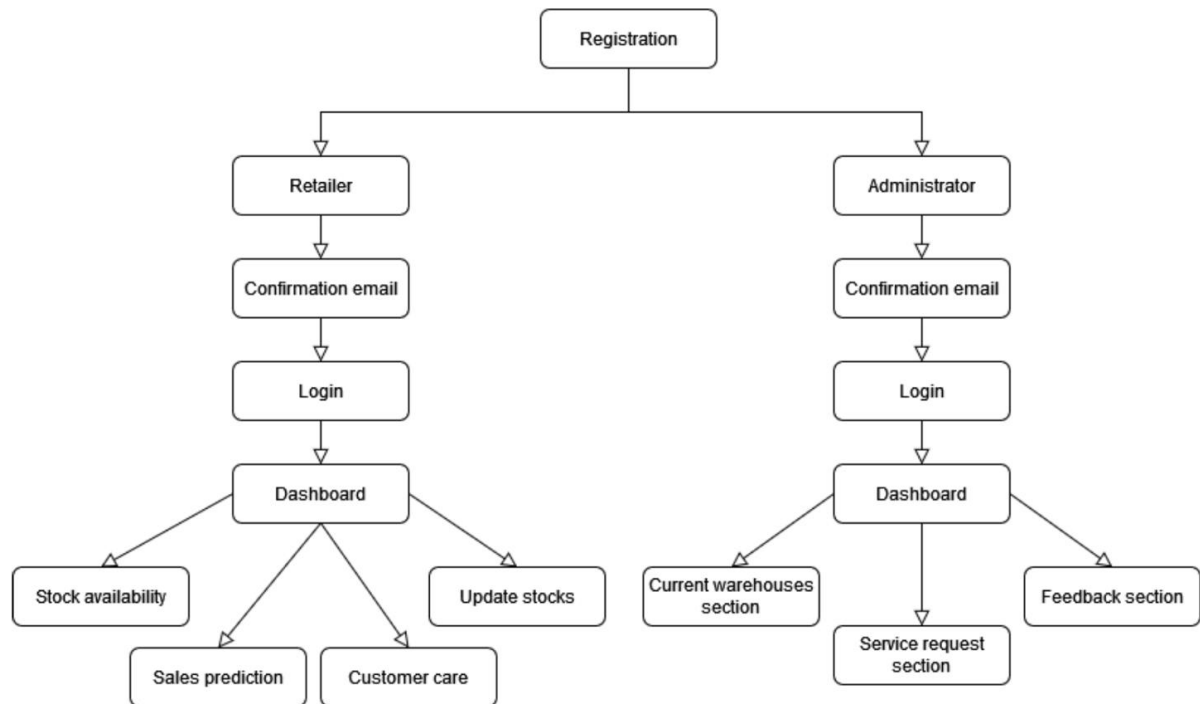
### 4.2 NON-FUNCTIONAL REQUIREMENTS:

FR No.	Non-Functional Requirement	Description
NFR-1	<b>Usability</b>	If the system has a steep learning curve, then it would mostly not be purchased by the company needing an inventory management system. <ul style="list-style-type: none"><li>• The UI is simple and easy to navigate</li><li>• Consistent design and colours are used.</li><li>• The webpages are responsive</li><li>• Email delivery is to be fast</li></ul>
NFR-2	<b>Security</b>	Security refers to the safety and management of the inventory of a company such that only authorised personnel are allowed to access them. <ul style="list-style-type: none"><li>• Login system is used to provide authentication.</li><li>• Users need to create account and verify it with their email OTP.</li><li>• Cookie based security is user for authentication on client side.</li></ul>

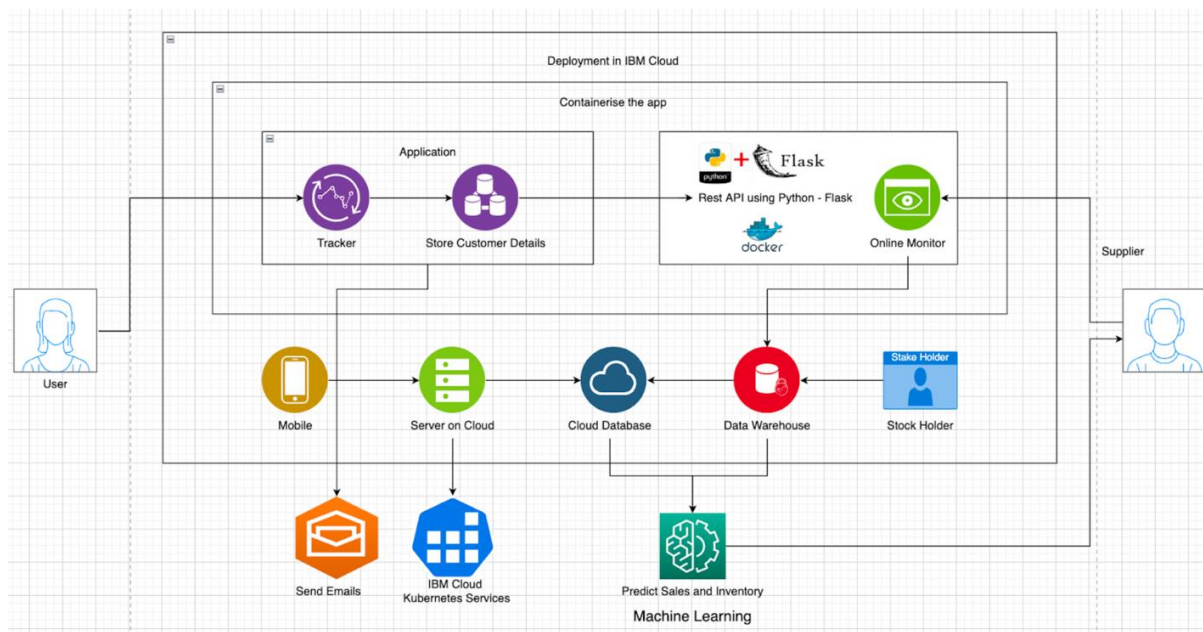
NFR-3	<b>Reliability</b>	<ul style="list-style-type: none"> <li>• Exception handling will be done at the code level to ensure that the app performs well even when errors happen in the runtime</li> <li>• Multiple instances of the App would be online to ensure continued operation</li> </ul>
NFR-4	<b>Performance</b>	<p>Performance of an inventory management system depends on the efficiency with which various tasks in it can be executed.</p> <ul style="list-style-type: none"> <li>• Reduces manpower, cost and saves time. Emails will be sent automatically when stocks are not available.</li> <li>• Makes the business process more efficient.</li> <li>• Improves organizations performance.</li> <li>• It will be perform fast and secure even at the lower bandwidth</li> </ul>
NFR-5	<b>Availability</b>	The use of IBM DB2 ensures high availability
NFR-6	<b>Scalability</b>	<p>The scalability of an inventory management system refers to the extensibility of its operations.</p> <ul style="list-style-type: none"> <li>• DB2 is highly Scalable</li> <li>• The code is developed efficiently to easily add new features without many changes by reusing the code.</li> <li>• Docker in IBM Container registry is used which is highly scalable</li> </ul>

## 5. PROJECT DESIGN

### 5.1. DATA FLOW DIAGRAMS



### 5.2. SOLUTION & TECHNICAL ARCHITECTURE



### 5.3. USER STORIES

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint -1
		USN-2	As a user, I can register for the application through E-mail.	I can access my account / dashboard	Medium	Sprint -1
	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the application.	I can get a confirmation for my email and password and create an authenticated account.	Medium	Sprint -1

	Login	USN-4	As a user, I can log into the application by entering the registered email & password.	I can log onto the application with the verified email and password	High	Sprint -1
	Dashboard	USN-5	As a user, I can view the products which are available.	Once I log on to the application, I can view the inventory.	High	Sprint -2
	Stock Update	USN-6	As a user, I can add products which are not available in the dashboard to the stock list.	If any of the products are not available, as a user I can update the inventory.	Medium	Sprint -2
	Sales Prediction	USN-7	As a user, I can get access to a sales prediction tool which will help me to better predict the order quantity.	The sales prediction tool should forecast the sales so that I, as a User, can order appropriately.	Medium	Sprint -3

Administrator	Request to Customer Care	USN-8	As a user, I am able to get in touch with the Administrator and ask for whatever services I require help with.	As a user, I can contact Customer Care and get support from them.	Low	Sprint -4
	Give feedback	USN-9	I should be able to report any difficulties I experience as a report.	As user, I can give my support in my possible ways to the administrator and to the administration.	Medium	Sprint -4

## 6. PROJECT PLANNING & SCHEDULING

### 6.1. SPRINT PLANNING & ESTIMATION

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and then confirming my password.	5	High	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-1		USN-2	As a user, I can register for the application through email.	3	Medium	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-1	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the application.	4	Medium	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-1	Login	USN-4	As a user, I can log into the application by entering the registered email & password	8	High	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-2	Dashboard	USN-5	As a user, I can view the products which are available.	10	High	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-2	Stock Update	USN-6	As a user, I can add products which are not available in the dashboard to the stock list.	10	Medium	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-3	Sales Prediction	USN-7	As a user, I can get access to a sales prediction tool which will help me to better predict the order quantity.	10	High	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-4	Administration	USN-8	As a user, I am able to get in touch with the Administrator and ask for whatever services I require help with.	10	Low	Harishankar. H, Arvind P, Nehanth K G, Premnaath V

Sprint-4		USN-9	I should be able to report any difficulties I experience to the administrator.	10	Medium	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
----------	--	-------	--	----	--------	--

## 6.2. SPRINT DELIVERY SCHEDULE

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	4 Nov 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	15 Nov 2022
Sprint-3	10	6 Days	07 Nov 2022	12 Nov 2022	10	22 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	25 Nov 2022

6.3.

## 6.3. REPORTS FROM JIRA

The screenshot shows the Jira Software interface for the project 'IMS for Retailers'. The left sidebar contains navigation options: PLANNING (Roadmap, Backlog), DEVELOPMENT (Board, Code), and Project pages (Project pages, Add shortcut, Project settings). The main area displays the 'Backlog' for the project. It shows three sprints: IFR Sprint 1 (15 Nov - 29 Nov, 4 issues), IFR Sprint 2 (15 Nov - 29 Nov, 2 issues), and IFR Sprint 3 (15 Nov - 29 Nov, 1 issue). Each sprint has a list of issues with their status (e.g., TO DO) and assigned users.



←

→

↺

🔒

🌐

🔗

📄

🏠

🔍

🌟

🔔

⚙️

👤

☰

https://inventoryibm.atlassian.net/jira/software/projects/IFR/boards/4

Jira Software

Your work

Projects

Filters

Dashboards

People

Apps

Create

🔍 Search

🔔

⚙️

👤

📌

IMS for Retailers

Software project

PLANNING

📅 Roadmap

📋 Backlog

**📊 Board**

DEVELOPMENT

🔧 Code

📄 Project pages

🔖 Add shortcut

⚙️ Project settings

You're in a team-managed project

Learn more

Projects / IMS for Retailers

**All sprints**

🔍

👤

👤

Sprint

🔗

🌟

Complete sprint

⋮

GROUP BY

None

📊 Insights

TO DO 9 ISSUES

IN PROGRESS

DONE ✓

+

dashboard to the stock list.

🟢 IFR-8

10

As a user, I can get access to a sales prediction tool which will help me to better predict the order quantity.

🟢 IFR-9

10

As a user, I am able to get in touch with the Administrator and ask for whatever services I require help with.

🟢 IFR-10

10

I should be able to report any difficulties I experience to the administrator.

🟢 IFR-11

10

IN PROGRESS 5 ISSUES

DONE 4 ISSUES ✓

As a user, I can add products which are not available in the dashboard to the stock list.

🟢 IFR-8

10

As a user, I can get access to a sales prediction tool which will help me to better predict the order quantity.

🟢 IFR-9

10

As a user, I am able to get in touch with the Administrator and ask for whatever services I require help with.

🟢 IFR-10

10

I should be able to report any difficulties I experience to the administrator.

🟢 IFR-11

10

As a user, I can register for the application through email.

🟢 IFR-2

3

As a user, I will receive confirmation email once I have registered for the application.

🟢 IFR-5

4

As a user, I can log into the application by entering the registered email & password.

🟢 IFR-6

8

Sprint progress

🔍

0% done

Done In progress Not started

0% 100% 0%

Sprint burndown

BETA

🔍

0 points done, 10 points to go

🔴 Heads up

100%

80%

60%

40%

20%

0%

Nov 15

Nov 29

● Remaining work

● Guideline

Epic progress

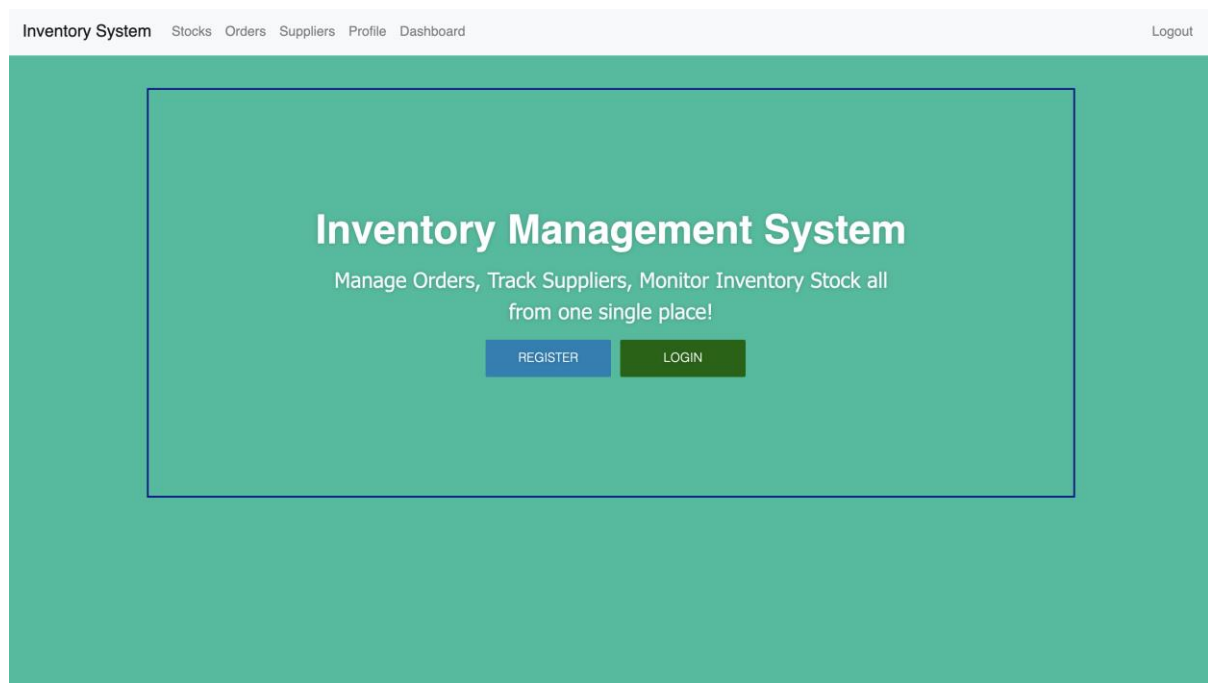
Link epics and estimate issues to drive your big goals

This insight shows how your current sprint contributes to your larger goals, or epics, helping to maintain focus and perspective. [Learn more](#)

## 7. CODING & SOLUTIONING (FEATURES ADDED ALONG WITH CODE)

### 7.1. FEATURE 1

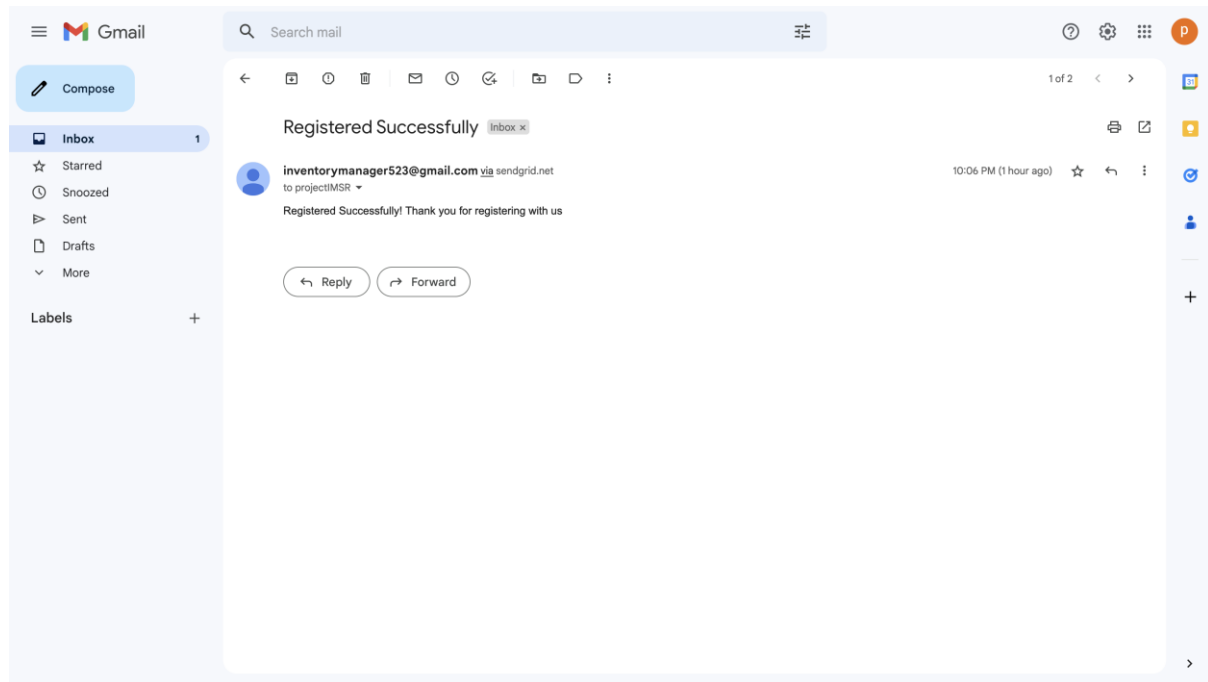
Used flask web framework to create an interactive dashboard



Users can register or login through this dashboard

## 7.2. FEATURE 2

Used SendGrid for autonomous emails



## 7.3. DATABASE SCHEMA (IF APPLICABLE)

### Tables

<input type="checkbox"/> Name ▼	Schema
<input type="checkbox"/> ORDERS	CQG39702
<input type="checkbox"/> STOCKS	CQG39702
<input type="checkbox"/> SUPPLIERS	CQG39702
<input type="checkbox"/> USERS	CQG39702

### Table definition

#### SUPPLIERS

No stati

Name	Data type	Nullable	Length	Scale
ORDER_ID	INTEGER	Y		0
SUPPLIER_ID	INTEGER	N		0
LOCATION	VARCHAR	Y	100	0
SUPPLIER_NAME	VARCHAR	Y	100	0

## 8. TESTING

### 8.1.TEST CASES

5	Test case ID	Feature Type	Component	Test Scenario	Pre-Requsite	Steps To Execute	Test Data	Expected Result	Actual Result	Status	Comments	TC for Automation(Yes/No)	BUG ID	Executed By
6	HomePage_TC_001	Functional	Home page	Verify user is able to move to the Register page to create a new account	-	1. Enter URL and click go 2. Click on the Register button / Click on the Register link in navbar	<a href="http://127.0.0.1">http://127.0.0.1</a>	User should be navigated to the Register page	Working as expected	Pass				Arvind P
7	HomePage_TC_002	Functional	Home page	Verify user is able to move to the Login page to create a new account	-	1. Enter URL and click go 2. Click on the Login button / Click on the Login link in navbar	<a href="http://127.0.0.1">http://127.0.0.1</a>	User should be navigated to the Login page	Working as expected	Pass				Arvind P
8	HomePage_TC_003	UI	Home Page	Verify the UI elements in the home page	-	1. Enter URL and click go 2. Verify that below UI elements exist: a. Register button b. Login button	<a href="http://127.0.0.1">http://127.0.0.1</a>	Page should show below UI elements: a. A blue Register button b. A green Login button	All mentioned elements exist	Pass				Arvind P
9	LoginPage_TC_001	Functional	Login page	Verify user is able to log into application with their correct credentials	Account must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password	<a href="http://127.0.0.1">http://127.0.0.1</a> Username: testusername Password: testpassword	Application should accept user credentials and user should be redirected to the dashboard	Working as expected	Pass				Neharsh K G
10	LoginPage_TC_002	Functional	Login page	Verify user is able to log into application with incorrect credentials	Incorrect account details must not exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password	<a href="http://127.0.0.1">http://127.0.0.1</a> Username: testusername Password: testpass	Application should show incorrect email or password - validation message	Working as expected	Pass				Neharsh K G
11	LoginPage_TC_003	UI	Login page	Verify the UI elements in the login page	-	1. Enter URL and click go 2. Verify that below UI elements exist: a. Username field b. Password field c. Login button d. Signup new link	<a href="http://127.0.0.1">http://127.0.0.1</a>	Page should show below UI elements: a. Username field b. Password field c. Login button d. Signup new link	All mentioned elements exist	Pass				Premnath V
12	RegisterPage_TC_001	Functional	Register page	Verify user is able to register to create a new account and get redirected to the login page	Account must not already exist	1. Enter URL and click go 2. Click on the Register button / Register link in navbar 3. Enter Name, Email, Username, Password, and Confirm Password	<a href="http://127.0.0.1">http://127.0.0.1</a> Name: testname Email: test@gmail.com Username: testusername Password: testpassword	New account should get created and user should be redirected to the login page	Working as expected	Pass				Premnath V
13	RegisterPage_TC_002	UI	Register page	Verify the UI elements in the login page	-	1. Enter URL and click go 2. Click on the Register button 3. Verify that below UI elements exist: a. Name field b. Email field c. Username field d. Password field e. Confirm Password field f. Submit button	<a href="http://127.0.0.1">http://127.0.0.1</a>	Page should show below UI elements: a. Name field b. Email field c. Username field d. Password field e. Confirm Password field f. Submit button	All mentioned elements exist	Pass				Premnath V
14	DashboardPage_TC_001	Functional	Dashboard page	Verify if the user can update stock, add new stock, and remove stock	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Stock Update: a. Enter item name, select a field and enter value. b. Click on Update 5. New Stock Addition: a. Enter item name, quantity, and price b. Click on Add Stock 6. Stock Removal: a. Enter item name b. Click on Remove	<a href="http://127.0.0.1">http://127.0.0.1</a> Item name: testitemname value:0 quantity:100 price:100	Stock gets updated, new stock gets added, and stock gets removed	Working as expected	Pass				Harishanker H
15	DashboardPage_TC_002	Functional	Dashboard page	Verify if the user cannot update non-existent stock, add already existing stock, or remove non-existent stock	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Stock Update: a. Enter invalid item name, select a field and enter value. b. Click on Update 5. New Stock Addition: a. Enter existing item name, quantity, and price b. Click on Add Stock 6. Stock Removal: a. Enter invalid item name b. Click on Remove	<a href="http://127.0.0.1">http://127.0.0.1</a> Item name: testitemname1 value:0 quantity:100 price:100	Respective error messages get shown	Working as expected	Pass				Harishanker H
16	DashboardPage_TC_003	UI	Dashboard page	Verify if the user is able to see a table of products along with functionality for updating, adding, and removing stock	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Dashboard page is displayed along with UI elements	<a href="http://127.0.0.1">http://127.0.0.1</a>	Page should show below UI elements: a. A table of products b. Three text fields and 'Update' button under Update Stock c. Three text fields and 'Add Stock' button under Add New Stock d. One text field and 'Remove' button under Remove Stock	All mentioned elements exist	Pass				Harishanker H
17	ProfilePage_TC_001	Functional	Profile page	Verify if the user can update their details and password	-	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Update user details and password	<a href="http://127.0.0.1">http://127.0.0.1</a> Username: testusername Password: testpassword	User details and password should get updated	Working as expected	Pass				Arvind P
18	ProfilePage_TC_002	UI	Profile page	Verify if the user is able to see their current details, and functionality to update their details and password	-	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Profile in navbar	<a href="http://127.0.0.1">http://127.0.0.1</a>	Page should show below UI elements: a. Current user details with username, name, and email b. Two text fields and 'Update' button under Update user details c. Three text fields and 'Update Password' button under Update Password	All mentioned elements exist	Pass				Arvind P
19	SuppliersPage_TC_001	Functional	Suppliers page	Verify if the user can update supplier, add new supplier, and delete supplier	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Suppliers in navbar 5. Supplier Update: a. Enter name, select a field and enter value. b. Click on Update 6. New Supplier Addition: a. Enter the supplier name, order ID, and location b. Click on Add Supplier 7. Supplier Deletion: a. Enter invalid supplier name b. Click on Delete	<a href="http://127.0.0.1">http://127.0.0.1</a> Supplier name: testsupplier location: abc Value: 100	Supplier details get updated, new supplier gets added, and a supplier is deleted	Working as expected	Pass				Neharsh K G

20	SuppliersPage_TC_002	Functional	Suppliers page	Verify if the user cannot update non-existent supplier, add already existing supplier, or remove non-existent supplier	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Suppliers in navbar 5. Supplier Update: a. Enter invalid name, select a field and enter value. b. Click on Update 6. New Supplier Addition: a. Enter invalid supplier name, order ID, and location b. Click on Add Supplier 7. Supplier Deletion: a. Enter invalid supplier name b. Click on Delete	<a href="http://127.0.0.1">http://127.0.0.1</a> Supplier name: testsupplier1 location: abc Value: 100	Respective error messages get shown	Working as expected	Pass				Neharsh K G
21	SuppliersPage_TC_003	UI	Suppliers page	Verify if the user is able to see a table of suppliers along with functionality for updating, adding, and deleting suppliers	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Suppliers in navbar	<a href="http://127.0.0.1">http://127.0.0.1</a>	Page should show below UI elements: a. A table of suppliers b. Two text fields, a dropdown, and 'Update' button under Update Supplier c. Two text fields, a dropdown, and 'Add Supplier' button under Add New Supplier d. One text field and 'Remove' button under Remove Supplier	All mentioned elements exist	Pass				Neharsh K G
22	OrdersPage_TC_001	Functional	Orders page	Verify if the user can create a new order, update an order, and cancel an order	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Orders in navbar 5. Order Creation: a. Enter Stock ID, quantity b. Click on Create 6. Order Update: a. Enter Order ID, choose a field, and enter value b. Click on Update 7. Order Cancellation: a. Enter Order ID b. Click on Cancel	<a href="http://127.0.0.1">http://127.0.0.1</a> Stock ID: 12345 Order ID: 123 Quantity: 100 Value: 100	New order gets created, an order gets updated, and an order gets cancelled	Working as expected	Pass				Harishanker H
23	OrdersPage_TC_002	Functional	Orders page	Verify if the user cannot update a non-existent order or cancel a non-existent order	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Orders in navbar 5. Order Update: a. Enter invalid Order ID, choose a field, and enter value b. Click on Update 6. Order Cancellation: a. Enter invalid Order ID b. Click on Cancel	<a href="http://127.0.0.1">http://127.0.0.1</a> Stock ID: 12345 Order ID: 124 Quantity: 100 Value: 100	Respective error messages get shown	Working as expected	Pass				Premnath V
24	OrdersPage_TC_003	UI	Orders page	Verify if the user is able to see a table of orders along with functionality for creating, updating, and cancelling orders	Table must exist	1. Enter URL and click go 2. Click on the Login button 3. Enter Username and Password 4. Click Orders in navbar	<a href="http://127.0.0.1">http://127.0.0.1</a>	Page should show below UI elements: a. A table of orders b. Two text fields and 'Create' button under Create Order c. Two text fields, a dropdown, and 'Update' button under Update Order d. One text field and 'Cancel' button under Cancel Order	All mentioned elements exist	Pass				Harishanker H

## 8.2.USER ACCEPTANCE TESTING

### 2. Defect Analysis

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	10	2	2	1	15
Duplicate	1	0	2	0	3
External	3	4	0	2	9
Fixed	7	2	1	15	25
Not Reproduced	0	0	2	0	2
Skipped	0	0	1	1	2
Won't Fix	0	3	2	1	6
Totals	21	11	10	20	62

### 3. Test Case Analysis

This report shows the number of test cases that have passed, failed, and untested

Outsource Shipping	1	0	0	1
Exception Reporting	6	0	0	6
Final Report Output	5	0	0	5
Version Control	2	0	0	2

## 9. RESULTS

### 9.1.PERFORMANCES METRICS

TEAM in	PNT2022TMD35555		NFT - Risk Assessment						
S.No	Project Name	Scope/feature	Functional	Hardware Changes	Software Changes	Impact of Downtime	Load/Volume Changes	Risk Score	Justification
1	Login	New	Moderate		Moderate		>30 to 50 %	ORANGE	Required feature
2	Transaction	New	High		Moderate		>30 to 50 %	RED	Indispensable feature
3	Containerizatio	New	Low		Moderate		>5 to 10%	ORANGE	Feature to make it deployable
NFT - Detailed Test Plan									
S.No	Project Overview		NFT Test approach		Assumptions/		Approvals/SignOff		
1	INVENTORY MANAGEMENT SYSTEM, GOB, RETAILER		Stress Test		Proper Internet Connection <i>(Note: Credentials)</i>		Approved		
End Of Test Report									
S.No	Project Overview	NFT Test approach	NFR - Met	Test Outcome	GO/NO-GO decision	Recommendations	Identified Defects (Detected/ Closed/Open)	Approvals/SignOff	

## 10. ADVANTAGES & DISADVANTAGES

### 10.1 ADVANTAGES

New	No Changes	No Changes	RED	In-Scope	Baseline Test
Existing	Low	>5 to 10%	ORANGE	Out of Scope	Benchmark
	Moderate	>10 to 30%	GREEN		
	High	>30 to 50 %			
		>50 to 70%			
		>70 to 100%			

- Used for small organizations
- Low stock alert as email

### 10.2 DISADVANTAGES

- This application is not suitable for those organizations where there is large quantity of product and different level of warehouses.
- This software application is able to generate only simple reports.
- Single admin panel is only made.
- It is not suitable for large organizations.

## 11. CONCLUSION

To conclude, Inventory Management System for retailers is a simple web-based application suitable for SMEs. It has all the necessities of a basic Inventory management system which are then used by organizations. Our team is successful in making the application where we can update, insert and delete the item as per the requirement. This application also sends an email alert when stock inventory is low. Though it has some limitations, our team strongly believes that the implementation of this system will surely benefit the organizations.

## 12. FUTURE SCOPE

Since this project was started with very little knowledge about the Inventory Management System, we came to know about the enhancement capability during the process of development. Some of the features which we can implement for the betterment and effectiveness of our project are listed below:

- Interactive user interface design.
- Manage Stock Godown wise.

## 13. APPENDIX

### 1. SOURCE CODE

#### app.py

```
from flask import Flask, render_template, flash, redirect, url_for, session, request, logging
from wtforms import Form, StringField, TextAreaField, PasswordField, validators, SelectField, IntegerField
import ibm_db
from functools import wraps
from datetime import datetime, timedelta
import sendgrid
import os
from sendgrid.helpers.mail import Mail, Email, To, Content

app = Flask(__name__)
app.secret_key = 'kekwcckqwodq'
#IBM DB2 Connection
```

```

try:
    conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=b0aebb68-94fa-46ec-a1fc-
1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud;PORT=31249;SECURIT
Y=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=cqg39702;PWD=hIRryoYSN
HJxjqQq", "", "")
except:
    print("Unable to connect: ", ibm_db.conn_error())

def sendgridmail(user,TEXT):
    sg = sendgrid.SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
    from_email = Email(os.environ.get('SENDGRID_FROM_EMAIL'))
    to_email = To(user)
    subject = "Registered Successfully"
    content = Content("text/plain",TEXT)
    mail = Mail(from_email, to_email, subject, content)

    # Get a JSON-ready representation of the Mail object
    mail_json = mail.get()
    # Send an HTTP POST request to /mail/send
    response = sg.client.mail.send.post(request_body=mail_json)
    print(response.status_code)
    print(response.headers)

@app.route('/')
def index():
    return render_template('home.html')

#Register Form Class
class RegisterForm(Form):
    name = StringField('Name', [validators.Length(min=1, max=50)])
    username = StringField('Username', [validators.Length(min=1, max=25)])
    email = StringField('Email', [validators.length(min=6, max=50)])
    password = PasswordField('Password', [
        validators.DataRequired(),
        validators.EqualTo('confirm', message='Passwords do not match')
    ])
    confirm = PasswordField('Confirm Password')

#user register
@app.route('/register', methods=['GET','POST'])
def register():
    form = RegisterForm(request.form)
    if request.method == 'POST' and form.validate():
        name = form.name.data
        email = form.email.data
        username = form.username.data
        password = str(form.password.data)

```



```

        sql = "SELECT * FROM users WHERE email=?"
        prep_stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(prepare_stmt, 1, email)
        ibm_db.execute(prepare_stmt)
        account = ibm_db.fetch_assoc(prepare_stmt)
        print(account)
        if account:
            error = "Account already exists! Log in to continue !"
        else:
            insert_sql = "INSERT INTO users (email,username,password,name)
values(?,?,?,?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(prepare_stmt, 1, email)
            ibm_db.bind_param(prepare_stmt, 2, username)
            ibm_db.bind_param(prepare_stmt, 3, password)
            ibm_db.bind_param(prepare_stmt, 4, name)
            ibm_db.execute(prepare_stmt)
            sendgridmail(email, "Registered Successfully! Thank you for
registering with us")
            flash(" Registration successful. Log in to continue !")

            #when registration is successful redirect to home
            return redirect(url_for('login'))
        return render_template('register.html', form = form)

#User login
@app.route('/login', methods = ['GET', 'POST'])
def login():
    if request.method == 'GET':
        return render_template('login.html')
    else:
        error = None
        account = None
        #Get form fields
        username = request.form['username']
        password = request.form['password']
        print(username, password)

        sql = "SELECT * FROM users WHERE username=? AND password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
    if account:
        session['logged_in'] = True
        session['username'] = username

```

```

        flash("Logged in successfully","success")
        return redirect(url_for('dashboard'))
    else:
        error = "Incorrect username / password"
        return render_template('login.html', error=error)

#Is Logged In
def is_logged_in(f):
    @wraps(f)
    def wrap(*args, **kwargs):
        if 'logged_in' in session:
            return f(*args, **kwargs)
        else:
            flash('Unauthorized, Please login', 'danger')
            return redirect(url_for('login'))
    return wrap

@app.route('/dashboard')
@is_logged_in
def dashboard():
    sql = "SELECT * FROM stocks"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    stocks = []
    print(dictionary)
    headings = [*dictionary]
    while dictionary != False:
        stocks.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)
    return render_template('dashboard.html',headings=headings, data=stocks)

@app.route('/logout')
@is_logged_in
def logout():
    session.clear()
    flash("Logged out successfully", "success")
    return redirect(url_for('login'))

@app.route('/inventoryUpdate', methods=['POST'])
@is_logged_in
def inventoryUpdate():
    if request.method == "POST":
        try:
            item = request.form['item']
            print("hello")
            field = request.form['input-field']
            value = request.form['input-value']
            print(item, field, value)

```

```

        insert_sql = 'UPDATE stocks SET ' + field + "= ?" + " WHERE
NAME=?"

        print(insert_sql)
        pstmt = ibm_db.prepare(conn, insert_sql)
        ibm_db.bind_param(pstmt, 1, value)
        ibm_db.bind_param(pstmt, 2, item)
        ibm_db.execute(pstmt)
        if field == 'PRICE_PER_QUANTITY' or field == 'QUANTITY':
            insert_sql = 'SELECT * FROM stocks WHERE NAME= ?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.execute(pstmt)
            dictionary = ibm_db.fetch_assoc(pstmt)
            print(dictionary)
            total = dictionary['QUANTITY'] *
dictionary['PRICE_PER_QUANTITY']
            insert_sql = 'UPDATE stocks SET TOTAL_PRICE=? WHERE NAME=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, total)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
    except Exception as e:
        msg = e

    finally:

        return redirect(url_for('dashboard'))

@app.route('/addstocks', methods=['POST'])
@is_logged_in
def addStocks():
    if request.method == "POST":
        print(request.form['item'])
        try:
            item = request.form['item']
            quantity = request.form['quantity']
            price = request.form['price']
            total = int(price) * int(quantity)
            insert_sql = 'INSERT INTO stocks
(NAME,QUANTITY,PRICE_PER_QUANTITY,TOTAL_PRICE) VALUES (?, ?, ?, ?)'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.bind_param(pstmt, 2, quantity)
            ibm_db.bind_param(pstmt, 3, price)
            ibm_db.bind_param(pstmt, 4, total)
            ibm_db.execute(pstmt)

        except Exception as e:

```

```

        msg = e

    finally:

        return redirect(url_for('dashboard'))

@app.route('/deletestocks', methods=['POST'])
@is_logged_in
def deleteStocks():
    if request.method == "POST":
        print(request.form['item'])
        try:
            item = request.form['item']
            insert_sql = 'DELETE FROM stocks WHERE NAME=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

    finally:
        return redirect(url_for('dashboard'))

@app.route('/update-user', methods=['POST', 'GET'])
@is_logged_in
def updateUser():
    if request.method == "POST":
        try:
            email = session['username']
            field = request.form['input-field']
            value = request.form['input-value']
            insert_sql = 'UPDATE users SET ' + field + '= ? WHERE username=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, email)
            print(pstmt)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)
            msg = e

    finally:
        if field == 'USERNAME':
            session['username'] = value
            return redirect(url_for('profile'))

@app.route('/update-password', methods=['POST', 'GET'])

```

```

@is_logged_in
def updatePassword():
    if request.method == "POST":
        try:
            email = session['username']
            password = request.form['prev-password']
            curPassword = request.form['cur-password']
            confirmPassword = request.form['confirm-password']
            insert_sql = 'SELECT * FROM users WHERE username=? AND
PASSWORD=?'

            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, email)
            ibm_db.bind_param(pstmt, 2, password)
            ibm_db.execute(pstmt)
            dictionary = ibm_db.fetch_assoc(pstmt)
            print(dictionary)
            if curPassword == confirmPassword:
                insert_sql = 'UPDATE users SET PASSWORD=? WHERE username=?'
                pstmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(pstmt, 1, confirmPassword)
                ibm_db.bind_param(pstmt, 2, email)
                ibm_db.execute(pstmt)
        except Exception as e:
            msg = e
        finally:

            return render_template('result.html')

@app.route('/orders', methods=['POST', 'GET'])
@is_logged_in
def orders():
    query = "SELECT * FROM orders"
    stmt = ibm_db.exec_immediate(conn, query)
    dictionary = ibm_db.fetch_assoc(stmt)
    orders = []
    headings = [*dictionary]
    while dictionary != False:
        orders.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)
    return render_template("orders.html", headings=headings, data=orders)

@app.route('/createOrder', methods=['POST'])
@is_logged_in
def createOrder():
    if request.method == "POST":
        try:
            stock_id = request.form['stock_id']

```

```

        query = 'SELECT PRICE_PER_QUANTITY FROM stocks WHERE ID= ?'
        stmt = ibm_db.prepare(conn, query)
        ibm_db.bind_param(stmt, 1, stock_id)
        ibm_db.execute(stmt)
        dictionary = ibm_db.fetch_assoc(stmt)
        if dictionary:
            quantity = request.form['quantity']
            date = str(datetime.now().year) + "-" + str(
                datetime.now().month) + "-" + str(datetime.now().day)
            delivery = datetime.now() + timedelta(days=7)
            delivery_date = str(delivery.year) + "-" + str(
                delivery.month) + "-" + str(delivery.day)
            price = float(quantity) * \
                float(dictionary['PRICE_PER_QUANTITY'])
            query = 'INSERT INTO orders
(STOCKS_ID,QUANTITY,DATE,DELIVERY_DATE,PRICE) VALUES (?, ?, ?, ?, ?)'
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, stock_id)
            ibm_db.bind_param(pstmt, 2, quantity)
            ibm_db.bind_param(pstmt, 3, date)
            ibm_db.bind_param(pstmt, 4, delivery_date)
            ibm_db.bind_param(pstmt, 5, price)
            ibm_db.execute(pstmt)
    except Exception as e:
        print(e)

    finally:
        return redirect(url_for('orders'))

@app.route('/updateOrder', methods=['POST'])
@is_logged_in
def updateOrder():
    if request.method == "POST":
        try:
            item = request.form['item']
            field = request.form['input-field']
            value = request.form['input-value']
            query = 'UPDATE orders SET ' + field + "= ?" + " WHERE ID=?"
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

    finally:
        return redirect(url_for('orders'))

```

```

@app.route('/cancelOrder', methods=['POST'])
@is_logged_in
def cancelOrder():
    if request.method == "POST":
        try:
            order_id = request.form['order_id']
            query = 'DELETE FROM orders WHERE ID=?'
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, order_id)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

        finally:
            return redirect(url_for('orders'))

@app.route('/suppliers', methods=['POST', 'GET'])
@is_logged_in
def suppliers():
    sql = "SELECT * FROM suppliers"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    suppliers = []
    orders_assigned = []
    headings = [*dictionary]
    while dictionary != False:
        suppliers.append(dictionary)
        orders_assigned.append(dictionary['ORDER_ID'])
        dictionary = ibm_db.fetch_assoc(stmt)

# get order ids from orders table and identify unassigned order ids
    sql = "SELECT order_id FROM orders"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    order_ids = []
    print("dictionary")
    print(dictionary)
    while dictionary != False:
        order_ids.append(dictionary['ORDER_ID'])
        dictionary = ibm_db.fetch_assoc(stmt)
    unassigned_order_ids=None

    # unassigned_order_ids = set(order_ids) - set(orders_assigned)
    return render_template("suppliers.html", headings=headings,
data=suppliers, order_ids=order_ids)

@app.route('/updatesupplier', methods=['POST'])

```

```

@is_logged_in
def UpdateSupplier():
    if request.method == "POST":
        try:
            item = request.form['name']
            field = request.form['input-field']
            value = request.form['input-value']
            print(item, field, value)
            insert_sql = 'UPDATE suppliers SET ' + field + "= ?" + " WHERE
NAME=?"

            print(insert_sql)
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))

```

```

@app.route('/addsupplier', methods=['POST'])
@is_logged_in
def addSupplier():
    if request.method == "POST":
        try:
            name = request.form['name']
            order_id = request.form.get('order-id-select')
            print(order_id)
            print("Hello world")
            location = request.form['location']
            insert_sql = 'INSERT INTO suppliers
(supplier_name,ORDER_ID,LOCATION) VALUES (?,?,?)'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, name)
            ibm_db.bind_param(pstmt, 2, order_id)
            ibm_db.bind_param(pstmt, 3, location)
            ibm_db.execute(pstmt)

        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))

```

```

@app.route('/deletesupplier', methods=['POST'])
@is_logged_in

```



```

def deleteSupplier():
    if request.method == "POST":
        try:
            item = request.form['name']
            insert_sql = 'DELETE FROM suppliers WHERE NAME=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))

@app.route('/profile', methods=['POST', 'GET'])
@is_logged_in
def profile():
    if request.method == "GET":

        email = session['username']
        insert_sql = 'SELECT * FROM users WHERE username=?'
        pstmt = ibm_db.prepare(conn, insert_sql)
        ibm_db.bind_param(pstmt, 1, email)
        ibm_db.execute(pstmt)
        dictionary = ibm_db.fetch_assoc(pstmt)
        print(dictionary)
        return render_template("profile.html", data=dictionary)

if __name__ == '__main__':

    app.run(host="127.0.0.1",port=5000,debug=True)

```

## 2. GITHUB AND PROJECT DEMO LINK

GitHub : <https://github.com/IBM-EPBL/IBM-Project-39229-1660401238>

Demo: <https://youtu.be/NiY1CfCZyao>

Deployed on Kubernetes: <http://169.51.205.124:32595>